Unisenac
Campus Pelotas

Agregação no MongoDB

Banco de Dados de Exemplo

Banco de Dados de Exemplo
Banco: empresa
Coleção: contatos
Objetivo: Aplicar agregações e visualizar contagem por tipo

Documentos:

```
db.contatos.insertMany([
  { nome: "João", tipo: "pessoal", telefone: "555-1234" },
  { nome: "Maria", tipo: "profissional", telefone: "555-5678" },
  { nome: "Pedro", tipo: "pessoal", telefone: "555-8765" },
  { nome: "Ana", tipo: "profissional", telefone: "555-4321" },
  { nome: "Paulo", tipo: "pessoal", telefone: "555-6789" }
])
```

Agregação no MongoDB

No SQL, você usaria um GROUP BY para agrupar dados e usar funções agregadas como COUNT, SUM, etc.

Exemplo em SQL:

```sql
SELECT tipo, COUNT(*) AS total
FROM contatos
GROUP BY tipo;
```

Resultado esperado:

tipo	total
pessoal	3
profissional	2

Agregação no MongoDB

MongoDB usa um "pipeline de agregação", semelhante a um encadeamento de etapas..

Exemplo em MongoDB:

```
db.contatos.aggregate([
  { $group: { _id: "$tipo", total: { $sum: 1 } } }
])
```

```
Resultado esperado:
[
  { "_id": "pessoal", "total": 3 },
  { "_id": "profissional", "total": 2 }
]
```

Agregação no MongoDB

O que cada parte faz:

```
{
  $group: {
    _id: "$tipo",       // agrupa por tipo
    total: { $sum: 1 } // conta quantos há em cada grupo
  }
}
```

Dica: também podemos somar valores reais

```
{ $group: { _id: "$departamento", totalSalario: { $sum: "$salario" } }
}
```

Operadores adicionais na agregação

Exemplo: Calcular média de salário no departamento "TI"

```
db.funcionarios.aggregate([
  { $match: { departamento: "TI" } },
  // filtra os documentos

  { $group: { _id: "$departamento", mediaSalario: { $avg: "$salario" } } },
  // agrupa e calcula média

  { $project: { _id: 0, departamento: "$_id", mediaSalario: 1 } }
  // renomeia e organiza a saída
])
```

Resultado:

```
{ "departamento": "TI", "mediaSalario": 7890.25 }
```

Agregação com ordenação e limite

Exemplo: Top 5 maiores salários (independente do departamento)

```
db.funcionarios.aggregate([
  { $sort: { salario: -1 } },
  { $limit: 5 },
  { $project: { _id: 0, nome: 1, salario: 1 } }
])
```

Operadores úteis

Operadores úteis para filtrar documentos

$in: valores dentro de uma lista
$exists: verifica se um campo existe
$type: verifica o tipo de dado do campo

Exemplo: Buscar funcionários do RH ou TI cujo campo salario existe:

```
db.funcionarios.find({
  departamento: { $in: ["TI", "RH"] },
  salario: { $exists: true, $type: "int" }
})
```

Validação de esquema

```
db.createCollection("funcionariosValidados", {
  validator: { // Define regras de validação
    $jsonSchema: {  // Usa a sintaxe JSON Schema para descrever como os documentos devem ser
      bsonType: "object", // O documento deve ser um objeto
      required: ["nome", "departamento", "salario"], // Campos obrigatórios
      properties: {   // Define regras específicas para cada campo
        nome: {
          bsonType: "string“  // Campo "nome" deve ser uma string
        },
        departamento: {
          enum: ["TI", "RH"] // Campo "departamento" deve ser exatamente "TI" ou "RH"
        },
        salario: {// Campo "salario" deve ser um número inteiro, no mínimo 3000
          bsonType: "int",
          minimum: 3000
        }
      }
    }
  }
})
```

Garante que os dados inseridos tenham os campos corretos.

Validação de esquema

`bsonType:` define o tipo esperado do campo (como string, int, object, etc.).
`required:` lista os campos obrigatórios.
`enum:` restringe os valores possíveis para um campo.
`minimum:` define o valor mínimo aceitável para campos numéricos.

Exemplo de documento válido:

```
{
  "nome": "Carlos",
  "departamento": "TI",
  "salario": 4500
}
```

Exemplo de documento inválido:

```
{
  "nome": "Ana",
  "departamento": "Financeiro",  // inválido! só aceita "TI" ou "RH"
  "salario": 2500                // inválido! menor que 3000
}
```

Índices no MongoDB

Por que usar índices?

Por que usar índices?

- Evitam escaneamento completo da coleção
- Aumentam a performance das consultas
- Essenciais em coleções com milhões de documentos
- Procuram documentos com base em critérios específicos

Gerando dados de teste

Banco: empresa
Coleção: funcionarios

```
for (let i = 0; i < 100000; i++) {
 db.funcionarios.insertOne({
  nome: `Funcionario${i}`,
  departamento: i % 2 === 0 ? "TI" : "RH",
  salario: Math.floor(Math.random() * 10000) + 3000
 });
}
```

100 mil documentos inseridos
Departamentos alternados (TI e RH)
Salário aleatório entre R$ 3.000 e R$ 13.000

Comparando Consultas: Sem Índices x Com Índices

Consulta sem índice

```
db.funcionarios.find({ departamento: "TI" }).explain("executionStats")
```

Resultado esperado: Full collection scan (lento)

Criar índice:

```
db.funcionarios.createIndex({ departamento: 1 })
```

Consulta com índice:

```
db.funcionarios.find({ departamento: "TI" }).explain("executionStats")
```

Resultado esperado: Index scan (rápido)

Desafio prático: MySQL vs MongoDB

1. Criar 500 mil registros em ambos os bancos:
   - empresa_sql com tabela funcionarios
   - empresa_nosql com coleção funcionarios

2. Executar as mesmas consultas (ex: WHERE departamento = 'TI’)

3. Comparar:
   - Tempo de execução
   - Total de registros examinados
   - Ganhos com e sem índices